TRANSPUTER PARALLEL PROCESSING AT NASA LEWIS RESEARCH CENTER

By

Graham K. Ellis
Institute for Computational Mechanics in Propulsion
NASA Lewis Research Center
Cleveland, Ohio

ABSTRACT

The transputer parallel processing lab at NASA Lewis Research Center (LeRC) consists of 69 processors (transputers) that can be connected into various networks for use in general purpose concurrent processing applications. The main goal of the lab is to develop concurrent scientific and engineering application programs that will take advantage of the computational speed increases available on a parallel processor over the traditional sequential processor.

Current research involves the development of basic programming tools. These tools will help standardize program interfaces to specific hardware by providing a set of common libraries for applications programmers.

The thrust of the current effort is in developing a set of tools for graphics rendering/animation. The applications programmer currently has two options for on-screen plotting. One option can be used for static graphics displays and the other can be used for animated motion.

The option for static display involves the use of 2-D graphics primitives that can be called from within an application program. These routines perform the standard 2-D geometric graphics operations in real-coordinate space as well as allowing multiple windows on a single screen. These real-coordinate routines can be used stand-alone for static displays or with the graphics engine, which is discussed below, for animated graphics.

For animation, a high-performance graphics engine has been developed. The graphics engine consists of 18 transputers connected in a 2-D mesh arrangement. Each node in the network performs a single graphics primitive computation. Frequently requested tasks such as line clipping can be performed on multiple nodes. The distribution of the normal display processor workload onto the distributed network increases the throughput by spreading the computationally intensive tasks over many processors.

The use of the graphics engine is transparent to the applications programmer other than requiring the inclusion of the appropriate pre-compiled code in the application program.

Future research targeted for the transputer lab includes parallelization of optimal and state-variable feedback control systems for simulating the control of both steady-state and transient vibration in rotor-dynamic systems.

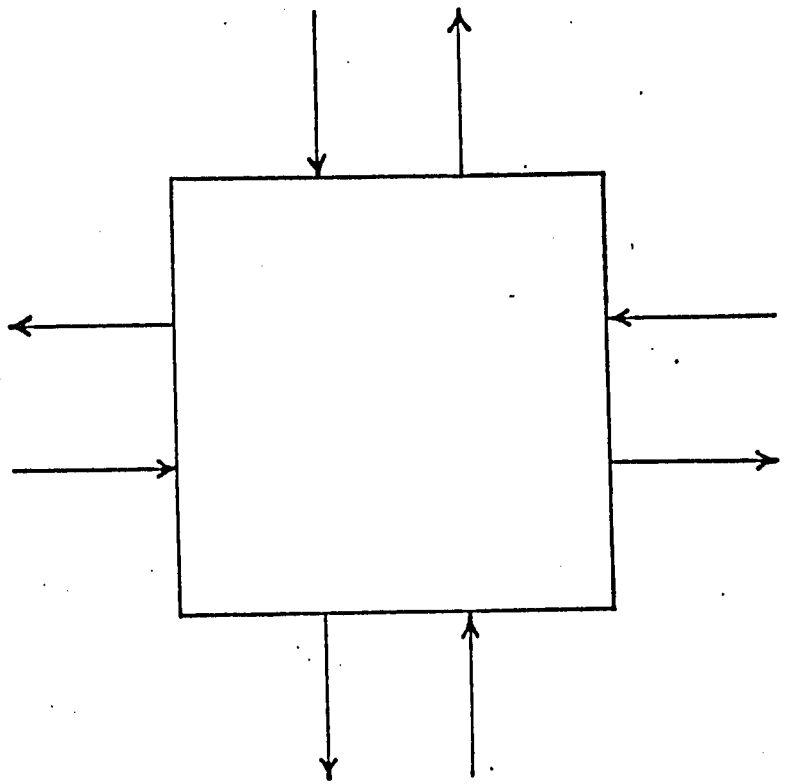
OVERVIEW

- Transputer hardware and software
- Graphics software development
- Future activities

A transputer is a microcomputer with its own local memory and links for connecting one transputer to another transputer.

A typical transputer contains a processor, memory and communication links on one chip.

The transputer can be used as a single chip processor or in networks to build high performance concurrent systems.



The transputer link arrangement.

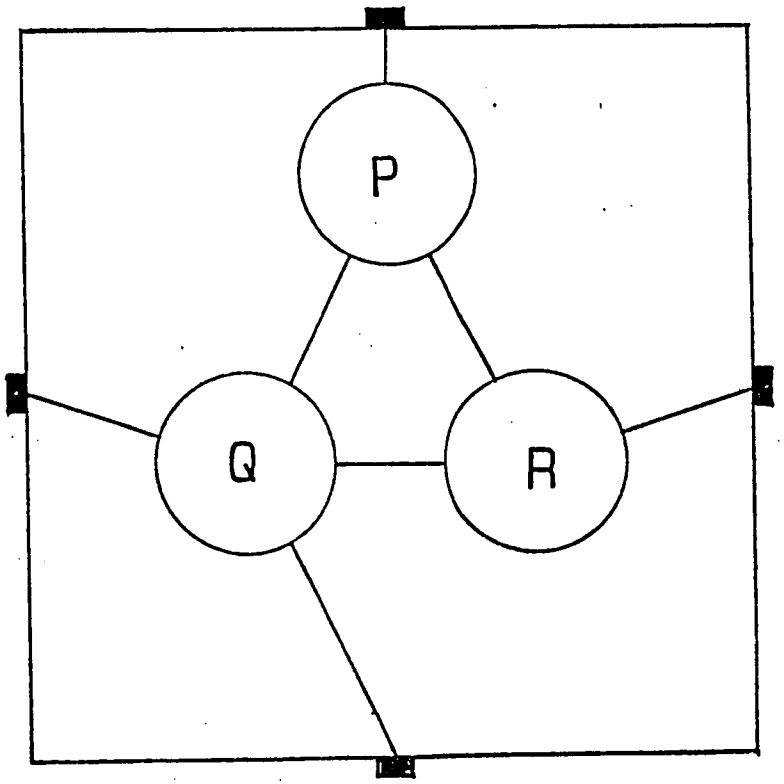
THIS PAGE LEFT BLANK INTENTIONALLY

The software building block is the process.

Each system is designed in terms of an interconnected set of processes.

Each process can be regarded as an independent unit of design. It communicates with other processes along point-to-point channels.

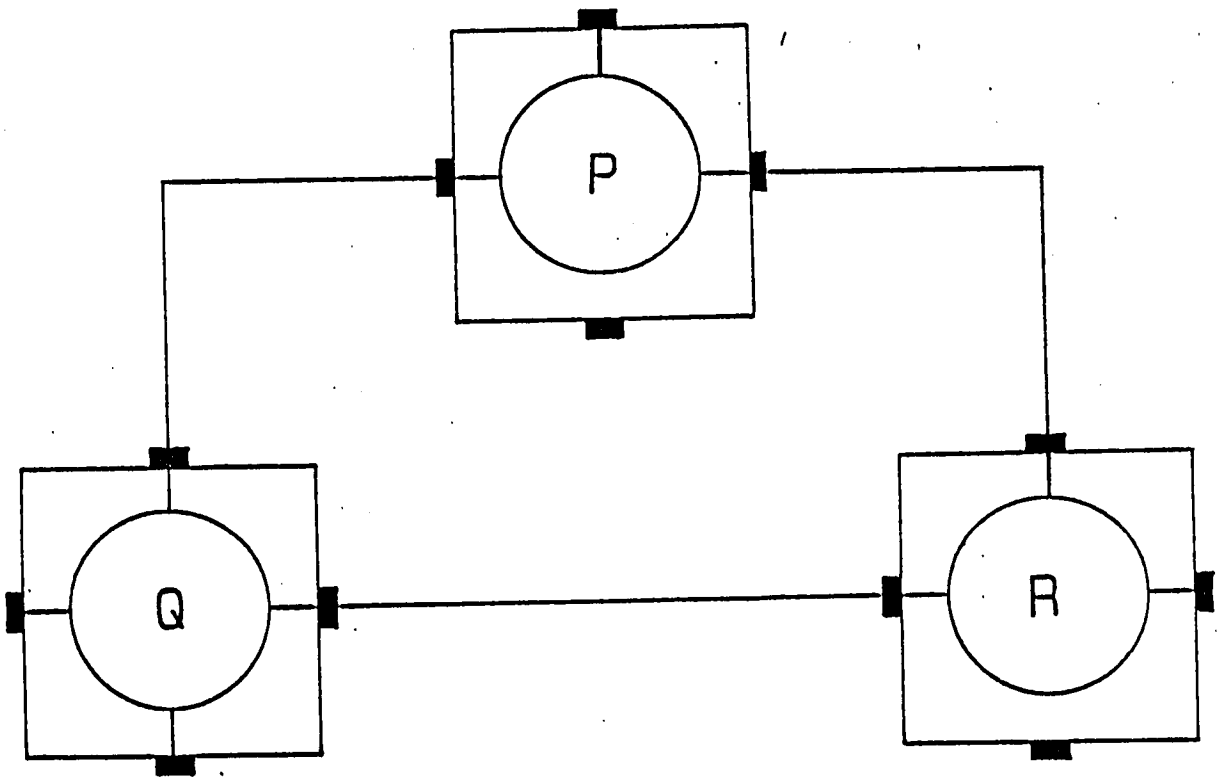
The transputer links are arranged as four pairs of input and output channels. These channel pairs allow the connection of large multi-transputer networks for parallel solution of computationally intensive problems.



A PROGRAM ON A SINGLE TRANSPUTER

A PROGRAM ON A SINGLE TRANSPUTER

The processes P, Q, and R are shown running in parallel (simulated with time-slicing) on a single transputer. The processes can only communicate with each other through channels. These channels are indicated by the lines connecting the processes. These channels are internal channels. They exist within the single transputer.



THE SAME PROGRAM ON THREE TRANSPUTERS

THE SAME PROGRAM ON THREE TRANSPUTERS

The processes P, Q, and R have now been distributed on a network of three transputers. These channels are physically placed on the links that connect the transputers in the network. Notice the lines representing the channels (and links in this case) connect the network in the same configuration shown above.

TRANSPUTER HARDWARE

- 40 T414 32-bit integer processors, 256KBytes per node (upgrade to T800 floating point chip 1/88)
- 27 T212 16-bit integer processors 24 with 8KBytes high-speed memory, 3 with 64KBytes
- 1 T414 based medium performance graphics board with 512x512 pixel resolution, 256 colors out of 256,000
- 1 T414 based development board, plugs into IBM PC slot, 2MBytes memory. System development software runs here

TRANSPUTER HARDWARE

The transputer lab hardware is as described here. The development system, a transputer-based card that plugs into an IBM compatible PC/XT or AT, is where the software development is performed. The other facilities are used only when performing an analysis or simulation.

OBJECTIVE

- Animate structural model vibration response

THIS PAGE LEFT BLANK INTENTIONALLY

APPROACH

- 2-D graphics primitive library for applications programs
- Distribute display processor workload on a network of transputers for increased throughput (graphics engine)

APPROACH

In order to generate plots, a library of 2-D routines for application programs has been developed. For animation, the 2-D routines above can be used with a graphics engine. This graphics engine consists of a network of 18 transputers on which the normal computational load of the display buffer is distributed.

2-D APPLICATION PRIMITIVES

- Graphics transformations
- Screen and window manipulation
- Relative coordinate commands
- Absolute coordinate commands

2-D APPLICATION PRIMITIVES

The following 2-D applications programs have been developed:

Graphics transformations

- scale
- rotate
- translate
- make.identity
- combine.transformations
- transform.points
- map.to.screen.coords

Screen and window manipulation

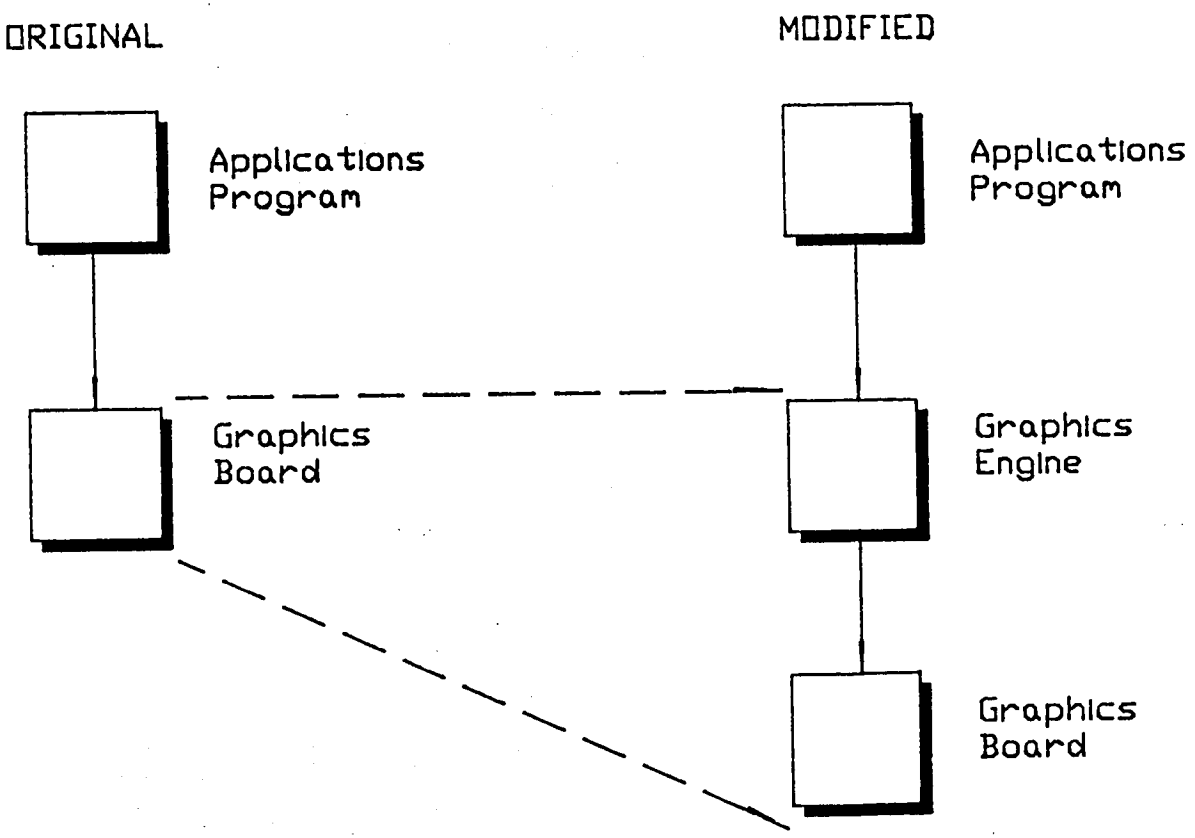
- set.viewport.2d
- activate.viewport.2d
- clip.line.2d
- clip.point.2d

Relative coordinate commands

- move.rel.2d
- point.rel.2d
- line.rel.2d

Absolute coordinate commands

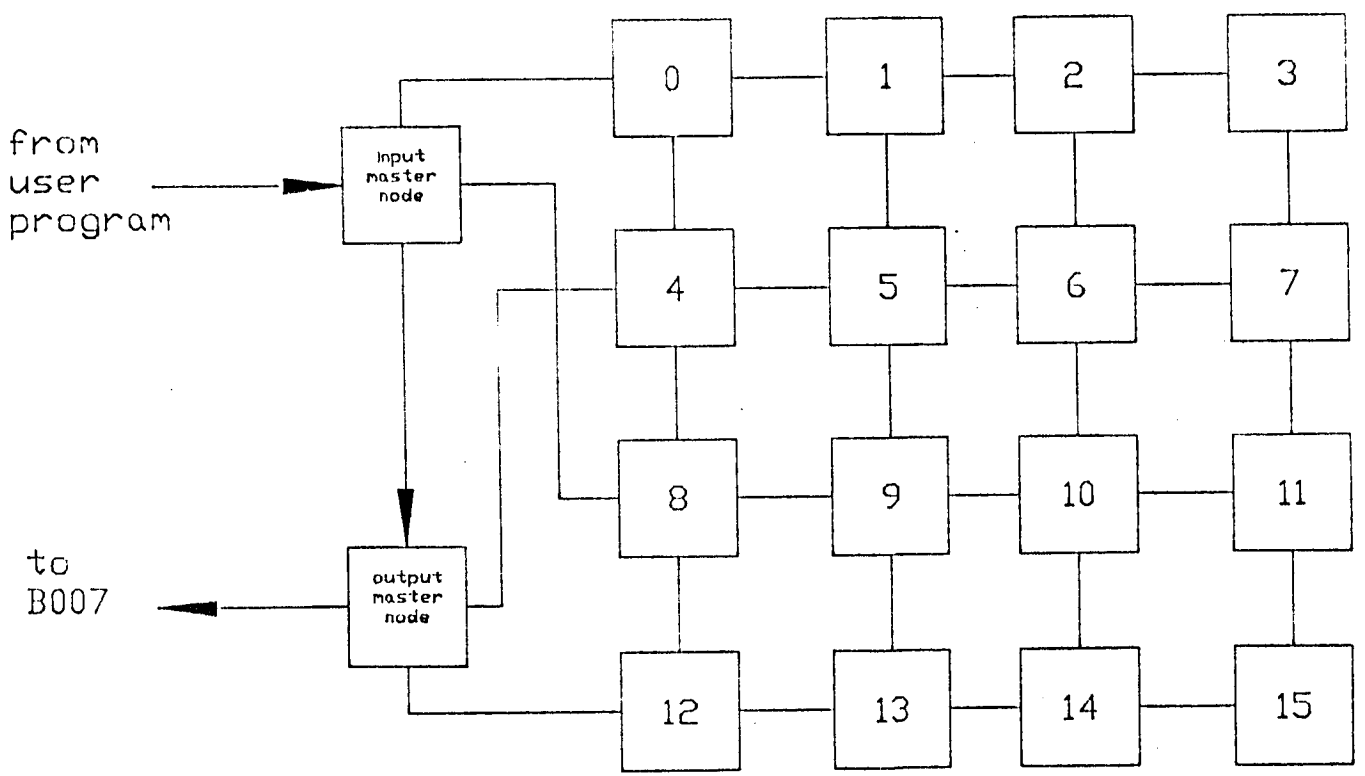
- move.abs.2d
- point.abs.2d
- line.abs.2d



Graphics Engine Implementation. Transparent to Applications Programs.

GRAPHICS ENGINE IMPLEMENTATION. TRANSPARENT TO APPLICATIONS PROGRAMS.

The use of the graphics engine is transparent to the applications programmer. The only difference is that the pre-compiled graphics engine code must be included in the applications program.



Multiple Processor Graphics Display Engine

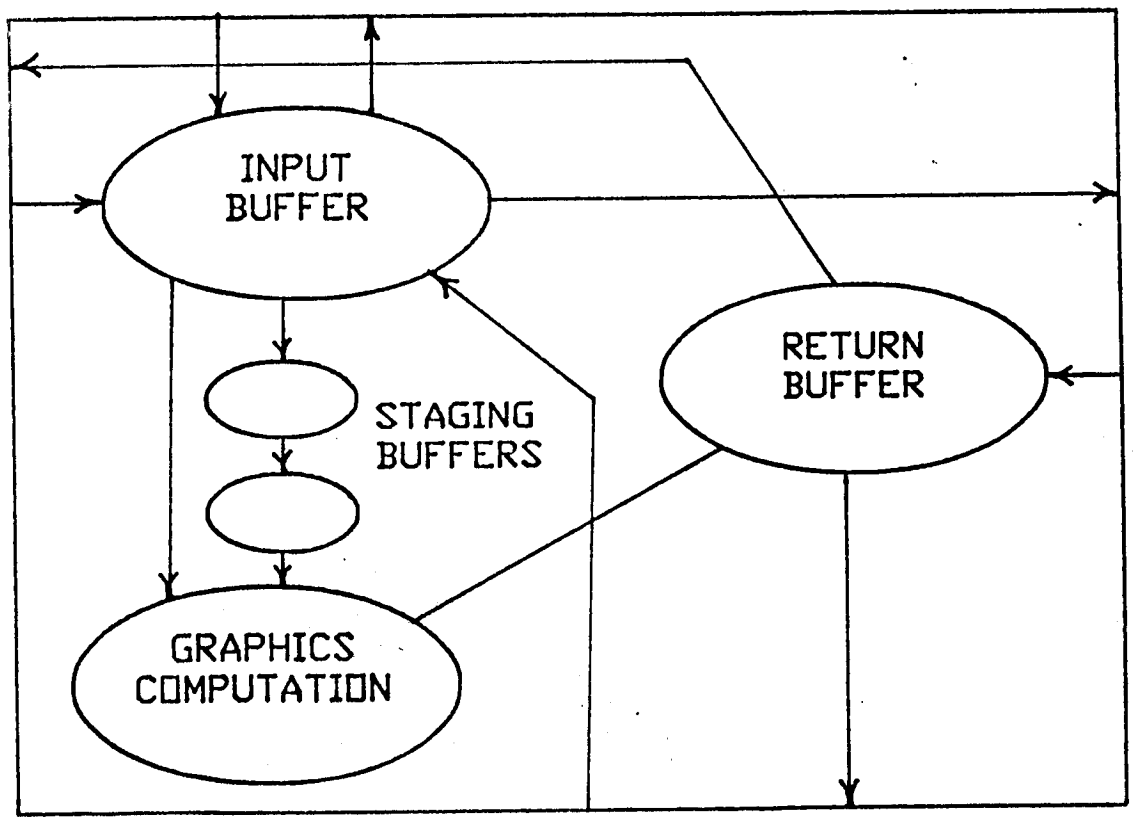
MULTIPLE PROCESSOR GRAPHICS DISPLAY ENGINE

The graphics engine consists of 18 20-MHz T212 16-bit integer transputers connected in a 2-D mesh arrangement. The 16 processors that make up the computational nodes (nodes 0 through 15) contain a total of 10 KBytes of memory including the 2 KBytes of transputer onchip memory. The input master and output master nodes each contain 64 KBytes of memory.

The input master node receives data from the users application program. A graphics command is decoded and data are either sent to the network if they are a graphics primitive operation or sent to the output master node if no processing is required by the graphics engine. The input master node also determines whether the requested graphics command entails the use of global variables. If global variables are required, the appropriate data are sent to each node in the network.

The output master node keeps track of the data received by the input master node. If the data are contained on the network, the output master sends a request for the pre-processed data to the appropriate node.

Each node performs only a single graphics primitive computation, such as line clipping and circle scan-line conversion. Frequently-requested commands such as line clipping can be performed on multiple nodes for a further increase in throughput.



GRAPHICS ENGINE NODE BUFFERS

GRAPHICS ENGINE NODE BUFFERS

Each node in the graphics engine contains several concurrently executing processes. The routing of data through the network is performed by the input and return buffers. The staging buffers store pending requests for the graphics computation process. The graphics primitive computation process performs its computation and waits for a signal to send the data back to the output master node through the return buffer.

Data routing decisions are made both in the input and return buffers.

KEY PROBLEMS

- How to handle global variables on a distributed network
- How to maintain computational synchronization on a distributed network
- How to avoid communication deadlock on the network

SOLUTIONS

- Each computational node has input and return buffers to intelligently communicate on the network
- All data sent on the network is tagged so the buffers can make the appropriate routing decisions
- Each node only performs a single graphics primitive computation. Frequently requested commands can reside on multiple nodes

SOLUTIONS

- Synchronization is maintained by the output master control node. It requests results from the network in the same order as the user application sent its requests for work

- Deadlock is avoided by using "staging buffers" on each node to store pending work requests. The input buffer controls the number of work requests sent to each node so the number of pending work requests is never larger than the number of staging buffers

FUTURE ACTIVITIES

- Parallel algorithms for optimal and state-variable feedback controls applications.
- Applications include control of unbalance forces in rotating machinery and control of transient vibrations

REFERENCES

1. Transputer Reference Manual, INMOS, Ltd., October 27, 1986.